**C++ Note**

1. **Basic I/O**

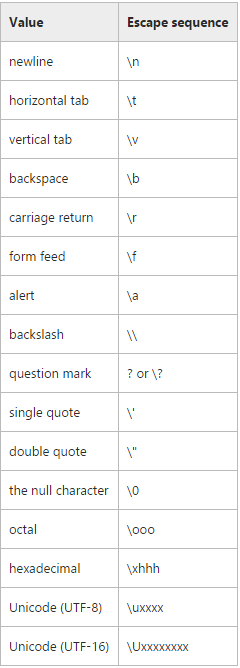
**1. Template**

|  |
| --- |
| **Basic program**  //header file for input/output function  #include <iostream>  using namespace std; //standard Library  int main(){  int a, b;  cin >> a >> b;  cout << "Sum a+b =" << a+b<< endl;  return 0;  } |
| **Read and write input from file** #include <iostream>  #include <fstream>  using namespace std;  int main()  {  ifstream input("./input.txt");  if (input.fail())  cout << "Failed to open input file!" << endl;  fstream output;  output.open("./output.txt", ios::out);    //Read line by line  //Get the first 255 characters then convert to string  while (!input.eof()) {  char char\_in[255];  input.getline(char\_in, 255);  string str = char\_in;  output << str << endl; //Write into outfile  }    input.close();  output.close();  system("pause"); //Pause the system to see the result  return 0;  } |
| **Use single library for all -> Only for Competitive programming**  //This includes all libraries we need, helps saving time (algorithm, iostream, vector,…)  #include <bits/stdc++.h>  using namespace std; |
| **Fast I/O Template :** freopen + cin/cout + unsync + \n  #include <bits/stdc++.h>  using namespace std;  const int MOD = 1e9 + 7;  int main() {  //This 2 lines help fasten I/O process 5.58s -> 0.48s  ios::sync\_with\_stdio(false);  cin.tie(nullptr);  //Read – Write file  freopen("speed.in", "r", stdin);  freopen("speed.out", "w", stdout);  int M, N;  cin >> M >> N;  int ans = 0;  for (int i = 0; i < N; i++) {  int x;  cin >> x;  ans = (ans + x) % MOD;  if (M == 1) { cout << ans << **"\n"**; } //use "\n" instead of endl  }  if (M == 0) { cout << ans << "\n"; }  } |

Header files for I/O

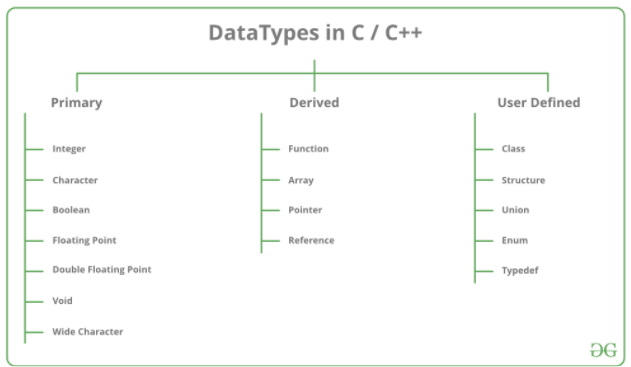
* **iostream**: cin, cout, cerr
* **iomanip:** iostream manipulator – để định dạng output  
  setw(int n) : setw(10) – set width là 10  
  setprecision(int n): setprecision(2) – set 2 chữ số thập phân trong kiểu float
* **fstream:** Handling files

**2. Keywords**

[cpp keywords links](https://en.cppreference.com/w/cpp/keyword)

|  |  |  |
| --- | --- | --- |
| **A – C** | **D – P** | **R – Z** |
| [alignas](https://en.cppreference.com/w/cpp/keyword/alignas) (C++11) [alignof](https://en.cppreference.com/w/cpp/keyword/alignof) (C++11) [and](https://en.cppreference.com/w/cpp/keyword/and) [and\_eq](https://en.cppreference.com/w/cpp/keyword/and_eq) [asm](https://en.cppreference.com/w/cpp/keyword/asm) [atomic\_cancel](https://en.cppreference.com/w/cpp/language/transactional_memory) (TM TS) [atomic\_commit](https://en.cppreference.com/w/cpp/language/transactional_memory) (TM TS) [atomic\_noexcept](https://en.cppreference.com/w/cpp/language/transactional_memory) (TM TS) [auto](https://en.cppreference.com/w/cpp/keyword/auto) (1) [bitand](https://en.cppreference.com/w/cpp/keyword/bitand) [bitor](https://en.cppreference.com/w/cpp/keyword/bitor) [bool](https://en.cppreference.com/w/cpp/keyword/bool) [break](https://en.cppreference.com/w/cpp/keyword/break) [case](https://en.cppreference.com/w/cpp/keyword/case) [catch](https://en.cppreference.com/w/cpp/keyword/catch) [char](https://en.cppreference.com/w/cpp/keyword/char) [char8\_t](https://en.cppreference.com/w/cpp/keyword/char8_t) (C++20) [char16\_t](https://en.cppreference.com/w/cpp/keyword/char16_t) (C++11) [char32\_t](https://en.cppreference.com/w/cpp/keyword/char32_t) (C++11) [class](https://en.cppreference.com/w/cpp/keyword/class) (1) [compl](https://en.cppreference.com/w/cpp/keyword/compl) [concept](https://en.cppreference.com/w/cpp/keyword/concept) (C++20) [const](https://en.cppreference.com/w/cpp/keyword/const) [consteval](https://en.cppreference.com/w/cpp/keyword/consteval) (C++20) [constexpr](https://en.cppreference.com/w/cpp/keyword/constexpr) (C++11) [constinit](https://en.cppreference.com/w/cpp/keyword/constinit) (C++20) [const\_cast](https://en.cppreference.com/w/cpp/keyword/const_cast) [continue](https://en.cppreference.com/w/cpp/keyword/continue) [co\_await](https://en.cppreference.com/w/cpp/keyword/co_await) (C++20) [co\_return](https://en.cppreference.com/w/cpp/keyword/co_return) (C++20) [co\_yield](https://en.cppreference.com/w/cpp/keyword/co_yield) (C++20) | [decltype](https://en.cppreference.com/w/cpp/keyword/decltype) (C++11) [default](https://en.cppreference.com/w/cpp/keyword/default) (1) [delete](https://en.cppreference.com/w/cpp/keyword/delete) (1) [do](https://en.cppreference.com/w/cpp/keyword/do) [double](https://en.cppreference.com/w/cpp/keyword/double) [dynamic\_cast](https://en.cppreference.com/w/cpp/keyword/dynamic_cast) [else](https://en.cppreference.com/w/cpp/keyword/else) [enum](https://en.cppreference.com/w/cpp/keyword/enum) [explicit](https://en.cppreference.com/w/cpp/keyword/explicit) [export](https://en.cppreference.com/w/cpp/keyword/export) (1) (3) [extern](https://en.cppreference.com/w/cpp/keyword/extern) (1) [false](https://en.cppreference.com/w/cpp/keyword/false) [float](https://en.cppreference.com/w/cpp/keyword/float) [for](https://en.cppreference.com/w/cpp/keyword/for) [friend](https://en.cppreference.com/w/cpp/keyword/friend) [goto](https://en.cppreference.com/w/cpp/keyword/goto) Bỏ [if](https://en.cppreference.com/w/cpp/keyword/if) [inline](https://en.cppreference.com/w/cpp/keyword/inline) (1) [int](https://en.cppreference.com/w/cpp/keyword/int) [long](https://en.cppreference.com/w/cpp/keyword/long) [mutable](https://en.cppreference.com/w/cpp/keyword/mutable) (1) [namespace](https://en.cppreference.com/w/cpp/keyword/namespace) [new](https://en.cppreference.com/w/cpp/keyword/new) [noexcept](https://en.cppreference.com/w/cpp/keyword/noexcept) (C++11) [not](https://en.cppreference.com/w/cpp/keyword/not) [not\_eq](https://en.cppreference.com/w/cpp/keyword/not_eq) [nullptr](https://en.cppreference.com/w/cpp/keyword/nullptr) (C++11) [operator](https://en.cppreference.com/w/cpp/keyword/operator) [or](https://en.cppreference.com/w/cpp/keyword/or) [or\_eq](https://en.cppreference.com/w/cpp/keyword/or_eq) [private](https://en.cppreference.com/w/cpp/keyword/private) [protected](https://en.cppreference.com/w/cpp/keyword/protected) [public](https://en.cppreference.com/w/cpp/keyword/public) | [reflexpr](https://en.cppreference.com/w/cpp/keyword/reflexpr) (reflection TS) [register](https://en.cppreference.com/w/cpp/keyword/register) (2) [reinterpret\_cast](https://en.cppreference.com/w/cpp/keyword/reinterpret_cast) [requires](https://en.cppreference.com/w/cpp/keyword/requires) (C++20) [return](https://en.cppreference.com/w/cpp/keyword/return) [short](https://en.cppreference.com/w/cpp/keyword/short) [signed](https://en.cppreference.com/w/cpp/keyword/signed) [sizeof](https://en.cppreference.com/w/cpp/keyword/sizeof) (1) [static](https://en.cppreference.com/w/cpp/keyword/static) [static\_assert](https://en.cppreference.com/w/cpp/keyword/static_assert) (C++11) [static\_cast](https://en.cppreference.com/w/cpp/keyword/static_cast) [struct](https://en.cppreference.com/w/cpp/keyword/struct) (1) [switch](https://en.cppreference.com/w/cpp/keyword/switch) [synchronized](https://en.cppreference.com/w/cpp/language/transactional_memory) (TM TS) [template](https://en.cppreference.com/w/cpp/keyword/template) [this](https://en.cppreference.com/w/cpp/keyword/this) (4) [thread\_local](https://en.cppreference.com/w/cpp/keyword/thread_local) (C++11) [throw](https://en.cppreference.com/w/cpp/keyword/throw) [true](https://en.cppreference.com/w/cpp/keyword/true) [try](https://en.cppreference.com/w/cpp/keyword/try) [typedef](https://en.cppreference.com/w/cpp/keyword/typedef) [typeid](https://en.cppreference.com/w/cpp/keyword/typeid) [typename](https://en.cppreference.com/w/cpp/keyword/typename) [union](https://en.cppreference.com/w/cpp/keyword/union) [unsigned](https://en.cppreference.com/w/cpp/keyword/unsigned) [using](https://en.cppreference.com/w/cpp/keyword/using) (1) [virtual](https://en.cppreference.com/w/cpp/keyword/virtual) [void](https://en.cppreference.com/w/cpp/keyword/void) [volatile](https://en.cppreference.com/w/cpp/keyword/volatile) [wchar\_t](https://en.cppreference.com/w/cpp/keyword/wchar_t) [while](https://en.cppreference.com/w/cpp/keyword/while) [xor](https://en.cppreference.com/w/cpp/keyword/xor) [xor\_eq](https://en.cppreference.com/w/cpp/keyword/xor_eq) |

1. **Data Types**



1. **Sizeof(), convert**

These are built-in or predefined data types. Can be used directly.

**Size of:**

cout << "Size of char : " << sizeof(char) << endl;

**Typeid:** Check type of

cout << "Type of x : " << typeid(x).name() << endl;

**Convert:**  
double x = 1.2

int y = (int)x + 3

**Constant:** (Biến Hằng số)  
const float gravity = 9.8;

**Define:** (Định nghĩa)  
#define PI 3.14159;

float area = PI \* (radius \* radius);

**Auto:** (Auto assign data type)  
auto age = 25;

auto max\_of\_int64 = INT64\_MAX;

auto character = 'V';

1. **Primary Data type**: Int, char, boolean

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Bytes** | **Value Range** |
| **char** | ASCII character or small integer. Numbers: 48 -> 57 (0-9) Characters: 65 -> 90 (A-Z)  97 -> 122 (a-z) | 1 | signed: -128 to 127 **unsigned**: 0 to 255 (ASCII) |
| **short int** (short) | Short integer | 2 | signed: -32768 to 32767 usigned: 0 to 65535  SHRT\_MIN / SHRT\_MAX |
| int | Integer | 4 | signed: -2 million to +2 million usigned: 0 to +4 million  INT\_MIN / INT\_MAX |
| **long int** (long) | Long integer | 4 | Same as above |
| **\_\_int16** |  | 2 | -32768 to 32767 |
| **\_\_int32** |  | 4 | -2mil ~ 2 mil |
| **\_\_int64** |  | 8 | -16 digits ~ 16digits |
| **bool** | True / False | 1 | **true** / **false** |
| **float** | Floating point number | 4 | +/- 3.4e +/- 38 (~7 digits)  FLT\_MIN / FLT\_MAX |
| **double** | Double precision floating point number | 8 | +/- 1.7e +/- 308 (~15 digits)  DBL\_MIN / DBL\_MAX |
| **long double** | Long double precision floating point number | 8 | +/- 1.7e +/- 308 (~15 digits) |
| **wchar\_t** | Wide character, to store Unicode text | 2 or 4 | 1 wide character |

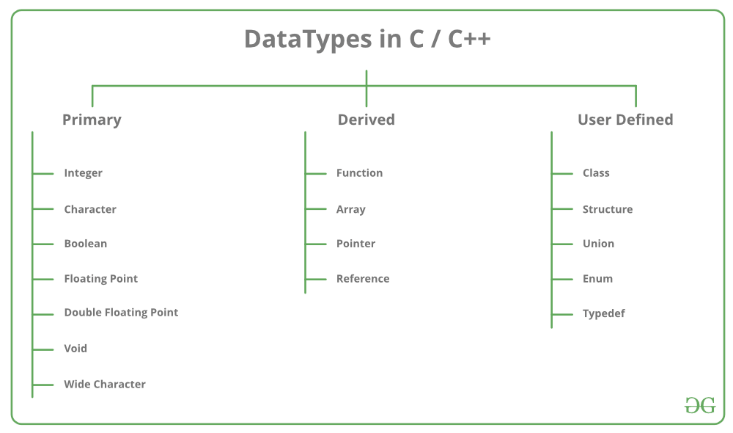
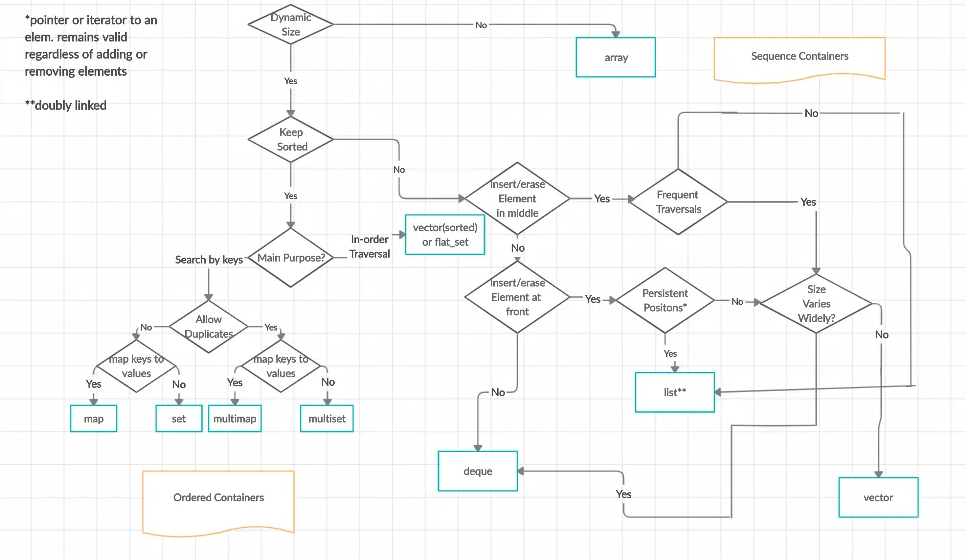
1. **Derived Data Types: function, array, pointer, reference**

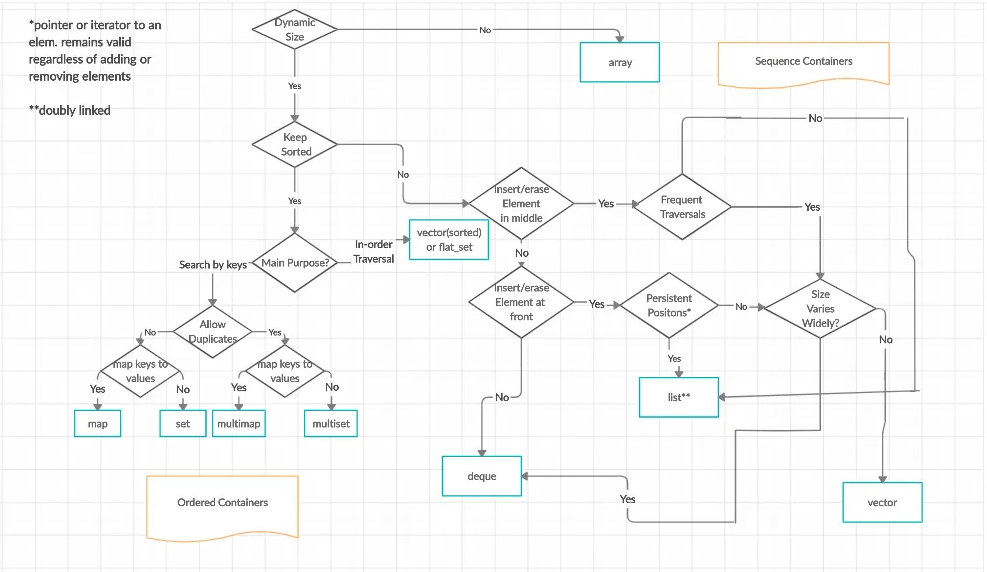
|  |  |  |
| --- | --- | --- |
| **Function** |  | int sum (int a, int b){  return a + b;  } |
| **Array** | collection of items stored at continuous memory locations. The idea of array is to represent many instances in one variable. | int arr[5]; arr[0] = 2;  char str[] = "Hello world. \n"; |
| **Pointer** | representation of address | int var = 20;  int**\*** pointer;  pointer = **&**var;  //value of pointer - address  cout<< pointer << "\n";  -> 0x7ffc10d7fd5c  //value of var cout<< var << "\n";  -> 20  //value stored in address  cout<< **\***pointer << "\n";  -> 20 |
| **Reference** | an alternative name for an existing variable  Tên khác | int x = 10;  int**&** ref = x;  //Value of x changed to 20  ref = 20; |

1. **Abstract or User-Defined Data Types: class, structure, union, Enum**

|  |  |  |
| --- | --- | --- |
| **Class** |  | using namespace std;  class **Geeks**  {  // Access specifier  **public**:  // Data Members  string geekname;    // Member Functions()  void printname()  {  cout << "Geekname is: " << geekname;  }  //Definition for Destructor  **~**Geeks()  {  cout << "Destructor called for id: " << id <<endl;  }  };    int main() {    // Declare an object of class geeks  **Geeks** obj1;    // accessing data member  obj1.geekname = "Abhi";    // accessing member function  obj1.printname();  return 0;  } |
| **Structure** |  |  |
| **Union** |  |  |
| **Enumeration** |  |  |
| **Typedef defined DataType** |  |  |

1. **Advanced Data Types**

****

****Adaptive container and unordered container

**0. Type Conversion**

// **Number to Number**

/\*(type)expression\*/

double x = 1.2;

int sum = (int)x + 0.5; //1 -> Round 1.7 down to 1

int sum2 = round(x + 0.5); //2 -> Round 1.7 up to 2

// **char to int**

char a = 'a'; //97

int num1 = (int)a; //97

char two = '2'; //50

int num2 = two - 48; //50-48 =2

// **int to char**

int num3 = 1;

char one = num3 + 48; //1 + 48 = 49 (ASCII code)

// **String to Number**

string s = "1.2345678";

int a1 = stoi(s);

long b = stol(s);

float c = stof(s);

double d = stod(s);

const char\* s2 = "12345";

int aa = atoi(s2);   
//Similar with stoi, but return 0 if string is null or not convertiable.

// **Number to string**

int num4 = 9999;

float f4 = 1.678;

string s4 = to\_string(num4); //Accept any number type

string sf4 = to\_string(f4);

// String to words / Split sentence to array

//Split string with ' '

string str = "Welcome to the computer world.";

vector<string> vec;

string word = "";

for (auto x : str)

{

if (x == ' ' or x == '.')

{

vec.push\_back(word);

word = "";

}

else

word = word + x;

}

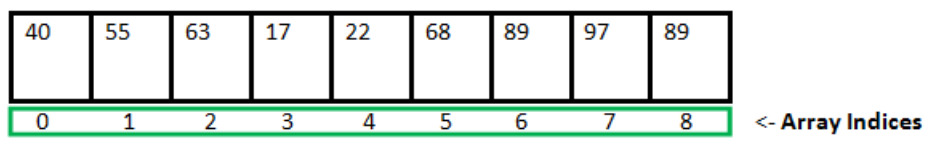
for (auto i : vec)

cout << i << endl;

**0.1 Pair & Tuple**

|  |  |
| --- | --- |
| **Pair - Hold 2 values**  pair<int, char> PAIR() {  // defining a pair  pair<int, char> cell;    //Access pair  cell.first = 100;  cell.second = 'G';  //Define a pair  pair<string, int> score = { "Hun", 80 };  //Define a pair using make\_pair()  pair<int, char> pair2 = **make\_pair**(200, 'A');  //Return a pair  return make\_pair(1, 'B');  }; | **Tuple - Hold more than 2 values**  tuple<string, string, int> TUPLE() {  int a = 3, b = 4, c = 5;  // defining a pair  tuple<int, int, int> t = tie(a, b, c);  cout << get<0>(t);  cout << get<1>(t);  cout << get<2>(t);  get<0>(t) = 7;  tuple<string, string, int> tp2 = **make\_tuple**("Hello", "world", 100);  //Return a tuple  return tp2;  } |

**1. Array**

Collection of items stored at continuous memory locations. The idea of array is to represent many instances in one variable.  
  


* The **size** of array is **fixed**

int arr[5];

arr[0] = 2;

char str[] = "Hello world. \n";

char my\_string[] = { 'H', 'e', 'l', 'l', 'o', '\0' };

**//get arr length**

int arrSize = sizeof(arr) / sizeof(arr[0])

#define ARRAY\_SIZE 100

**//main**

float f\_values[ARRAY\_SIZE];

**2D array**

int array2D[3][5]; // 3x5 elements (3 rows, 5 columns)

int array2D[3][5] =

{

{ 1, 2, 3, 4, 5 }, //row 1

{ 6, 7, 8, 9, 10 }, //row 2

{ 11, 12, 13, 14, 15 } //row 3

};

int seats[3][5] = //assign **missing elements**

{

{ 1, 2 }, //row 1 = 1, 2, 0, 0, 0

{ 6, 7, 8 }, //row 2 = 6, 7, 8, 0, 0

{ 11 }, //row 3 = 11, 0, 0, 0, 0

};

int array2D[][4] = //assign empty row

{

{ 1, 2, 3, 4 },

{ 5, 6, 7, 8 }

};

//2D array travelsal

int board[3][4];

for (int row = 0; row < 3; row++)

{

for (int col = 0; col < 4; col++)

{

cout << board[row][col] << " ";

}

cout << endl;

}

### **Visited 2D array**

#include <stdio.h>

#include <iostream>

using namespace std;

#define ROW 3

#define COL 5

int main() {

bool visited[ROW][COL];

for (int row = 0; row < ROW; row++)

{

for (int col = 0; col < COL; col++)

{

visited[row][col] = false;

}

}

system("pause");

return 0;

}

**2. Vector / Dynamic array**

#include <vector>

vector<int> vec = {1, 2, 3};

// fill the array with 12 seven times

vec.**assign**(7, 12);

// Size / Length

vec.**size**();

// inserts 24 to the last position

vec.**push\_back**(24);

vec.**emplace\_back**(20);

// inserts 10 at the beginning

vec.**insert**(vec.begin(), 10);

vec.**emplace**(vec.begin(), 10);

// removes last element

vec.**pop\_back**();

// removes the first element

vec.**erase**(vec.begin());

// removes the by index

vec.**erase**(vec.begin() + index);

// erases the vector

vec.**clear**();

**Iterator / For loop**

Is an object that can **iterate over elements** in a C++ standard library container and **provide access** to individual elements. -> Là vòng lặp / con trỏ để access 1 phần tử.

vec.**begin**() //The 1st element

vec.**end**() //The last element

|  |
| --- |
| for (auto i = vec.**begin**(); i != vec.**end**(); ++i)  cout << \*i << " "; |

for (auto i : vec)

cout << i << " ";

**Sorting**

#include <vector>

#include <algorithm>

**//sort(start,end)**

**//Increase array**

int arr[6] = { 5, 4, 3, 2, 1, 0 };

sort(arr, arr + 6); // 0 1 2 3 4 5

vector<int> vec = { 5, 4, 3, 2, 1, 0 };

sort(vec.begin(), vec.end());

string str = “bbccaaadd”;

sort(str.begin(), str.end()); // aabbccdd

**//Decrease array**

bool comp(const int a, const int b) {

return a > b;

}

int arr[6] = { 0, 1, 2, 3, 4, 5 };

sort(arr, arr + 6, comp); // 5 4 3 2 1 0

vector<int> vec = { 0, 1, 2, 3, 4, 5 };

sort(vec.begin(), vec.end(), comp); // 5 4 3 2 1 0

### **Vector 2D array**

|  |
| --- |
| vector<vector<int>> v{ {1, 0, 1},  {0, 1},  {1, 0, 1}  };  for (int i = 0;i < v.size();i++) {  for (int j = 0;j < v[i].size();j++)  cout << v[i][j] << " ";  cout << endl;  } |

### **Visited-Vector 2D array**

|  |
| --- |
| //inline  vector<vector<bool>> visited2(ROW, vector<bool>(COL, false)); //DONE  //OR  vector<vector<bool>> visited2;  for (int i = 0; i < numRows; i++) {  //Add empty {} every new Row  visited2.push\_back({});  for (int j = 0; j < numRows; j++) {  visited2[i].push\_back(false);  }  }    //iterating through 2D vector  for (int i = 0;i < ROW;i++) {  for (int j = 0;j < COL;j++)  cout << visited2[i][j] << " ";  cout << endl;  } |
| 0 0 0 0  0 0 0 0  0 0 0 0 |

**3. Strings**

#include <iostream>

#include <string>

using namespace std;

#define MAX\_LENGTH 50

int main() {

char c1[MAX\_LENGTH];

char c2[] = "Hello World";

char c3[] = { 'H', 'l','l','o',' ', 'W','o', 'r', 'l','d','\0'};

//Get data from input

char full\_name[MAX\_LENGTH];

cout << "Enter your full name: "; //Will Smith

cin >> full\_name; //full\_name = {'Will\0'}

//Get data with white space

cin.**getline**(full\_name, MAX\_LENGTH); //Need MAX LENGTH

**gets\_s**(full\_name);

//Get length

cout << c2.size() << endl;

//Compare 2 strings

if (**strcmp**(c2, c3) == 0) //return 0 if equal

cout << "str1 and str2 are equal" << endl;

else

cout << "str1 ans str2 are not equal" << endl;

//Find target in a string

char text[] = "This is a simple string";

char target[] = "simple";

char\* pos = strstr(text, target); //pos: position index

if (pos == NULL)

cout << "Could not find the pattern string in the text string" << endl;

else {

int32\_t match\_index = (pos - text) / sizeof(char);

cout << "The pattern string match the text string at: " << match\_index << endl;

}

//-------------------------------------------------------

// string assignment

string s1 = "Hello";

string s2 = "World";

string str;

//Getline with white space

cout << "Please enter your name: \n";

getline(cin, str);

// Get length

cout << s1.**size**() << endl; // 5

cout << s2.**length**() << endl; // 5

// Concert to string

// concatenate string using + operator.

s1 = s1 + s2;

cout << s1 << endl; // HelloWorld

// append string

s1.**append**("Geeks");

s1 += "Hello";  
 s1.**pop\_back**(); // Remove last character

string s3 = "HelloWorldGeeks";

// compare two strings

if (s1.**compare**(s3) == 0) //return 0 if equal

cout << "true" << endl;

else

cout << "false" << endl;

// substring of string s1

// substr(pos, length\_of\_substring)

string sub = s1.substr(0, 5);

cout << sub << endl; // Hello

// insert into string

// insert(pos, string)

s1.insert(10, "For");

cout << s1 << endl; // HelloWorldForGeeks

// find a target string in s1

string target = "World";

size\_t pos = s1.**find**(target);

if (pos != std::string::npos) // npos=-1

cout << "Found at Position:" << pos << endl; // pos=5

// replace a portion of string s1

// replace(pos, length\_of\_portion, string\_to\_replace)

cout << s1.replace(5, 5, "Geeks") << endl; // HelloGeeksForGeeks

return 0;

}

// Copy two characters of s1 (starting

// from position 3)

string r = s1.substr(3, 2);

**3.1 Checking character type**

string str0 = "This is an expression: 1 + 1/2 = 1.5";

string filtered\_string; // Remove all non-character, non-number

for (auto ch : s) {

if (isalnum(ch))

filtered\_string += tolower(ch);

}

isalnum(c); // is Alphabet - Number

isalpha(c); // is Alphabet

isdigit(c); // is Digit Number

isblank(c); // is Blank Space '\t'

isspace(c); // is White Space ' '

islower(c); ->tolower(c); // abc

isupper(c); -> toupper(c); // ABC

**4. Stack - Queue**

**LILO FIFO Vector**

|  |  |  |
| --- | --- | --- |
| #include <stack>  **stack<int> s;**  s.**push**(1);  **int** top = s.**top**();  s.**pop**();  if (!s.**empty**()) | #include <queue>  **queue<int> q;**  q.**push**(1);  //get 1st element  **int** front = q.**front**();  //get Last element **int** back = q.**back**();  q.**pop**();  if (!q.**empty**()) | #include <vector>  **vector<int>** vec = {};  vec.**push\_back**(1);  vec.push\_back(2);  vec.push\_back(3);  int front = vec.**front**();  int back = vec.**back**();  vec.**pop\_back**(); //remove last  vec.**erase**(vec.**begin**()); //Remove first  if (!vec.**empty**())  cout << vec.**size**() << endl;  vec.**clear**(); |

**5. Heap – Priority Queue**

A set of elements that are in non-decreasing order (or priority queue).

|  |
| --- |
| #include <queue>  using namespace std;  // driver code  void test()  {  int arr[6] = { 10, 2, 4, 8, 6, 9 };  //**Max\_heap**  **priority\_queue**<int> max\_queue;  for (auto i : arr){  max\_queue.**push**(i);  }  while (!max\_queue.**empty**()) {  cout << max\_queue.**top**() << ' ';  max\_queue.**pop**();  }  //10 9 8 6 4 2  //---------------------------------------------------------------------  //**Min\_heap**  **priority\_queue<int, vector<int>, greater<int>>** min\_queue;  for (auto i : arr) {  min\_queue.**push**(i);  }  //2 4 6 8 9 10  //---------------------------------------------------------------------  //Priority queue of pairs (Ordered by first)  //frequency, value  priority\_queue<pair<int, int>> pq;  } |

**6. Hash map**

**6.1 map vs unordered\_map vs multimap**

|  |  |  |
| --- | --- | --- |
|  | **map** | **unordered\_map** |
| **Header**  **Implementation**  **Order**  **Search Time**  **Insertion Time**  **Deletetion Time** | #include <map>  [key]:value  Red-Black Tree – Self balancing BST  Increasing order - Sorted (by default)  log(n) -> **Slow**  Log(n) + rebalance  Log(n) + rebalance | #include <unordered\_map>  [key]:value  Hash Table  Unordered  O(1) -> Average -> **Fast** O(n) -> Worst case  O(1)  O(1) |
|  | #include <iostream>  #include <map>  int main()  {  // Ordered map  std::**map**<int, int> order;  // Mapping values to keys  order[5] = 10;  order[3] = 5;  order[20] = 100;  order[1] = 1;  // Iterating the map and  // printing ordered values  for (auto i : order){  std::cout << i.**first**  << " : "  << i.**second** << '\n';  }  } | #include <iostream>  #include <unordered\_map>  int main()  {  // Ordered map  std::**unordered\_map**<int, int> order;  // Mapping values to keys  order[5] = 10;  order[3] = 5;  order[20] = 100;  order[1] = 1;  // Iterating the map and  // printing ordered values  for (auto i : order){  std::cout << i.**first**  << " : "  << i.**second** << '\n';  }  } |
|  | 1 : 1  3 : 5  5 : 10  20 : 100 | 1 : 1  20 : 100  5 : 10  3 : 5 |

**Multimap**Similar to map but **1 key** can directs to **multiple values.**  
**[key] : n-values**

|  |
| --- |
| #include <iostream>  #include <map>  using namespace std;  multimap<int, int> multimap;  multimap.insert(pair<int, int>(1, 60));  multimap.insert(pair<int, int>(**2, 50**));  multimap.insert(pair<int, int>(**2, 10**)); |

**6.2 map**

#include <iostream>

#include <map>

using namespace std;

int main()

{

// Create a map of strings to integers

map<string, int> map;

// Insert some values into the map

map["one"] = 1;

map["two"] = 2;

map["three"] = 3;

// Get an iterator pointing to the first element in the map

std::map<string, int>::iterator it = map.**begin**();

// Iterate through the map and print the elements

while (it != map.**end**())

{

cout << "Key: " << it->first << ", Value: " << it->second << endl;

++it;

}

// Print the size of the map

cout << "Size of map: " << map.size() << endl;

// Check if a key is in the map

if (map.**count**("four") > 0)

{

cout << "Key 'four' is in the map" << std::endl;

}

//Remove map['two']

map.**erase**("two");

//Removes all

map.**clear**();

return 0;

}

1. **User define data types**

**1. Structure**

Used to **group** items of **different types** **into a single type.**

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  struct **Person** {  \_\_int32 ID;  string name;  \_\_int16 age;  float height;  float weight;  bool isMale;  void PrintInfo() {  //print all info  }  };  int main() {  //Initialization  **Person Chang** = { 123, "Chang Dang", 22, 159.5, 50, false };  Chang.height = 175;  //Create a list of students  **vector<Person> students;**  students.**push\_back**(Chang); //Add a Person  students.push\_back({456, "Hun", 25, 165.0, 60, true}); //Add a Person  students[1].isMale = false; //Access an element  //Iterating through a struct  for (auto stu : students) {  cout << "Name: " << stu.name << " ID: "<< stu.ID<< "\n";  }  system("pause");  return 0;  } |

**2. Union**

To define a group of dif data types into 1 type. **BUT** **all member share the same memory location.**   
-> Change 1 member will change other members.

|  |
| --- |
| union test {  int x, y;  };  // int main(){  union test uni;  uni.x = 2; // uni.y = 2  uni.y = 10; // uni.x = 10 |

**3. Enum**

Enumeration (or enum) is used to assign names to intergral constants. -> Make **code easier to read and maintain**.

|  |
| --- |
| enum State { Working = 1, Failed = 0, Sleep = 0 };  //Enum vs Define : Same same  #define Working 0  #define Failed 1  #define Sleep 0  enum Direction  {  UP = 1, //assigned 1 by programmer  DOWN = 3, //assigned 3 by programmer  LEFT, //assigned 4 by compiler  RIGHT //assigned 5 by compiler  };  //start counting fromt 0 to 6.  enum week { Mon, Tue, Wed, Thur, Fri, Sat, Sun };  // int main(){  enum week day1, day2, day3;  day1 = week::Wed; //day1 = 2  day2 = Fri; //day2 = 4  day3 = 6; //ERROR  switch (day1)  {  case Mon:  //do something  break;  case Tue:  //do something  break;  default:  break;  } |

**4. Class**

**5. Typedef**

1. **Memory**

**1. Reference &**

An **alternative name** for an existing variable. **Both share the same memory.** (Virtual memory)

|  |
| --- |
| int x = 1;  int**&** ref = x; //reference of x  //address of x, Same with ref  cout << **&**x << endl; //00EFFB64  cout << **&**ref << endl; //00EFFB64  x = 5; //ref = 5  ref = 10; //x = 10  //ref can only point to x, can’t reset or be NULL |
| 1. **Modify the passed paremeters in a function.**  void pass\_string(string**&** message, ) {  message += " Nice to meet you.";  } |
| 2. **Avoid create a copy of a large structure. Pass a parameter will create a clone of variable.**  void pass\_struct(**const** Student**&** person) {  cout << person.name << person.ID << person.age << endl;  } |
| **3. Modify all objects in FOR EACH LOOP**  vector<int> vect{ 10, 20, 30, 40 };  // We can modify elements if we  // use reference  for (int**&** x : vect) {  x = x + 5;  }  // Printing elements  for (int x : vect) {  cout << x << " ";  }  //15 25 35 45  } |

**2. Pointer \***

Are used to **store the address** of variables or memory location.

-> Provides **low-level memory access**  
**-> Dynamic memory allocation**

|  |
| --- |
| int x = 1;  int**&** ref = x; //reference of x  int**\*** ptr = **&**x; //pointer stores x's address  int\* ptr2 = **new** int;  //address of x, Same with ref  cout << **&**x << endl; //00EFFB64  cout << **&**ref << endl; //00EFFB64  cout << pointer << endl; //00EFFB64  //The address stored in pointer **can’t be** changed. Only point to x's address  **const** int**\*** const\_ptr = &x;  //**Wild pointer** - stores random value  char\* str;  char\* str2 = **new** char;  //**NULL pointer** - to avoid wild pointer.  char\* str = **NULL**;  //**Array** pointer  int arr[3] = {1, 2, 3};  int**\*** arr\_ptr = arr; //No Need &  int**\*** arr\_ptr2 = **new** int[3];  //**Struct** pointer  Student**\*** str\_ptr;  Student**\*** str\_ptr2 = **new** Student();    //Free memory space  **delete** ptr;  **delete** arr\_ptr;  **delete** str\_ptr;  **delete** pointer\_double; |
| **Multilevel pointers** **- Store other pointer’s address**  //Double pointer  int**\*\*** pointer\_double = **&**pointer;  //Multilevel pointer  int**\*\*\*** pointer\_triple = **&**pointer\_double; |

**3. Smart Pointer**

Helps **automate free memory** allocation process. (Don’t need to call **delete** )  
[Smart Pointer in C++](https://www.youtube.com/watch?v=UOB7-B2MfwA&ab_channel=TheCherno)

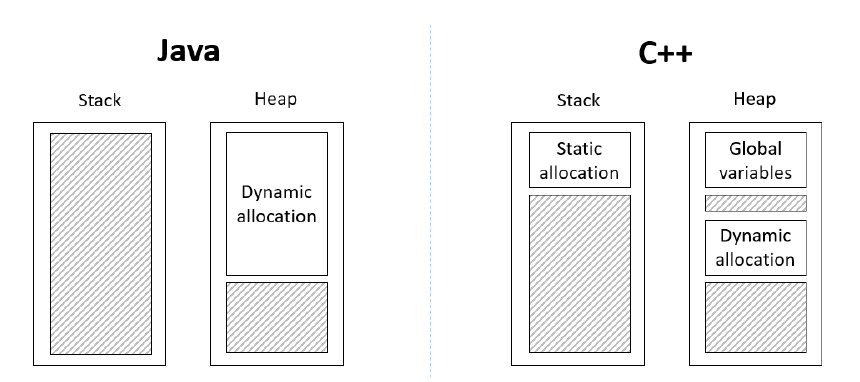
**std::unique\_ptr**

**stf::shared\_ptr**

**std::weak\_ptr**

**4. Stack vs Heap memory**

[**[Youtube]Stack vs Heap Memory in C++**](https://www.youtube.com/watch?v=wJ1L2nSIV1s&ab_channel=TheCherno)



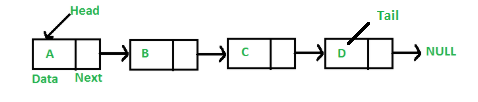
**Memory allocation** (Phân phát bộ nhớ): **Stack and Heap** that are areas where we can **store data.**

|  |  |  |
| --- | --- | --- |
| Both stack and heap memory are in RAM | | |
| Parameter | **Stack** | **Heap** |
| Type of data structures | A stack is a **linear data** structure. | Heap is a hierarchical data structure. |
| Memory Allocation | Memory is allocated in a **contiguous block.** | Memory is allocated in **any random order.** |
| Access speed | **Faster** | **Slower** |
| Space management | Space managed efficiently by OS so memory will never become fragmented. | Heap Space not used as efficiently. Memory can become fragmented as blocks of memory first allocated and then freed. |
| Access | **Local variables only** | It allows you to **access variables globally.** |
| Allocation and Deallocation | **Automatically** done by compiler instructions. | It is **manually** done by the programmer. |
| Deallocation | Does not require to de-allocate variables. | Explicit de-allocation is needed. |
| Cost | Less | More |
| Implementation | A stack can be implemented in 3 ways simple array based, using dynamic memory, and Linked list based. | Heap can be implemented using array and trees. |
| Flexibility | **Fixed size** | **Resizing is possible** |
| Limit of space size | Limit on stack size | Does not have a specific limit on memory size. |
| Resize | Variables cannot be resized | can be resized. |
| Main Issue | **Shortage of memory** | **Memory fragmentation** |
| Possible Error | **Stack overflow** | **Memory Overwrite Out Of Memory Error** |
| **Advantages** | Local variables in function are stored in stack, it’s automatically clear() once returned.  -> **Automatically free** memory when the code goes out of current scope.  int stack = 5;  int array\_stack[3];  - Not easily corrupted  - Used for fast access | Can have full control over memory located   -> **Manually free** memory by “**delete**”, or it will be there until the memory blocks are **overwritten**.  “**new**” – to dynamically allocate memory  “malloc()” – “free()” are similar with new  – delete but in C libraries.  int\* heap = new int;  \*heap = 5; int\* array\_heap = new int[3];  delete heap;  delete[] array\_heap;  - Randomly and globaly access  - Used to store big variables |

1. **Linked List**

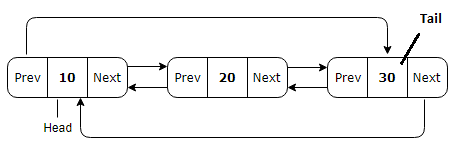
**1. Single Linked list**

- Linked list is a linear data structure, elements in linked list are linked using pointer.   
- **The size of linked list** is **flexible** while the size of array is fixed. -> Dynamic size.   
- Random access is not allowed. We have to access elements from the first node to the last node.

**  
  
Time Complexity**

|  |  |  |  |
| --- | --- | --- | --- |
| Access | Search | Insert | Delete |
| O(n) | O(n) | O(1) | O(1) |

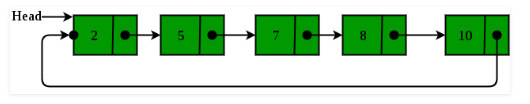
**2. Double Linked List (DLL)**

****

**3. Circular Linked list**

**Circular linked list***is a linked list where all nodes are connected to form a circle. There is* ***no NULL*** *at the end. ->* ***No tail****.   
- Any node can be a head*

**-** Useful for implementation of queue

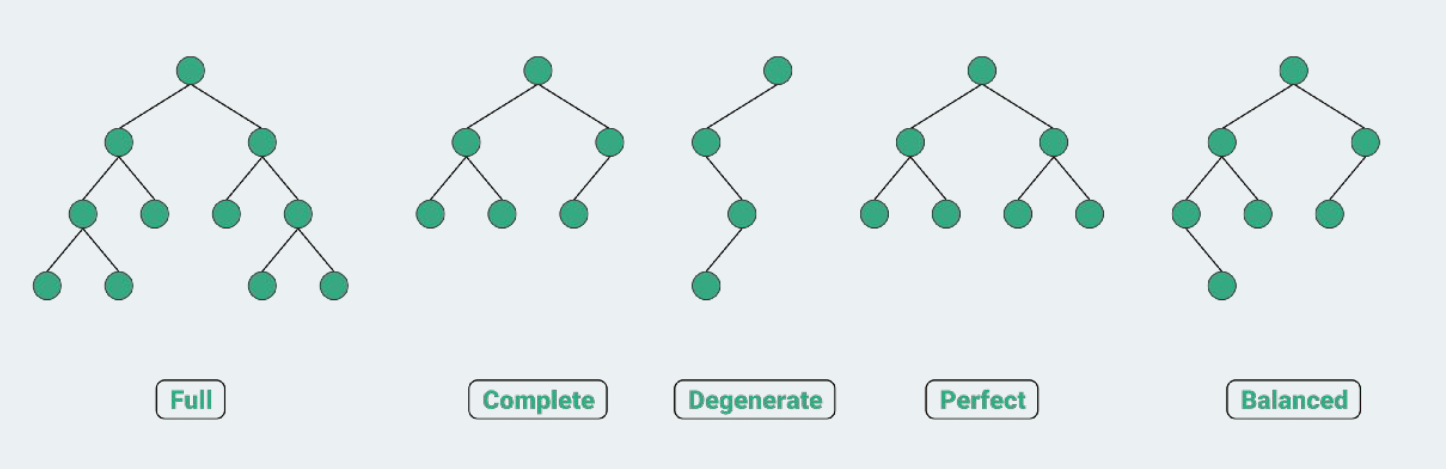


1. **Tree**

**1. Tree Basic**

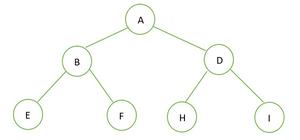
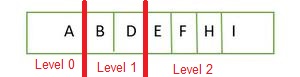
A tree data structure is a **hierarchical structure** that is used to represent and organize data in a way that is **easy to navigate and search**.

Some Types of trees:

* **Balanced tree**: Left sub-tree and Right sub-tree have the same height or different 1 height.

**2. Binary Tree**

A tree has most 2 children: Left or Right node.  
A binary tree with **height h** -> maximum number of nodes: **2h+1-1**  
Maximum number of nodes at level n: **2n**  
Access element stored in array:  
root: **a[0]**  
parent: **a[n]**  
left-child: **a[2n+1]**  
right-child: **a[2n+2]**

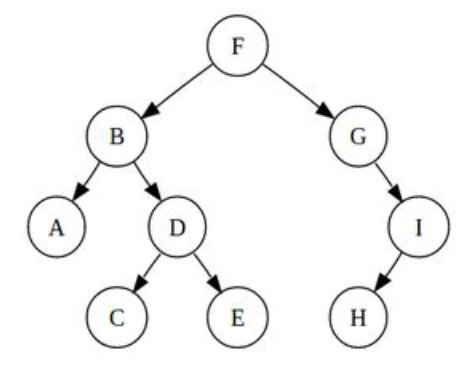
 

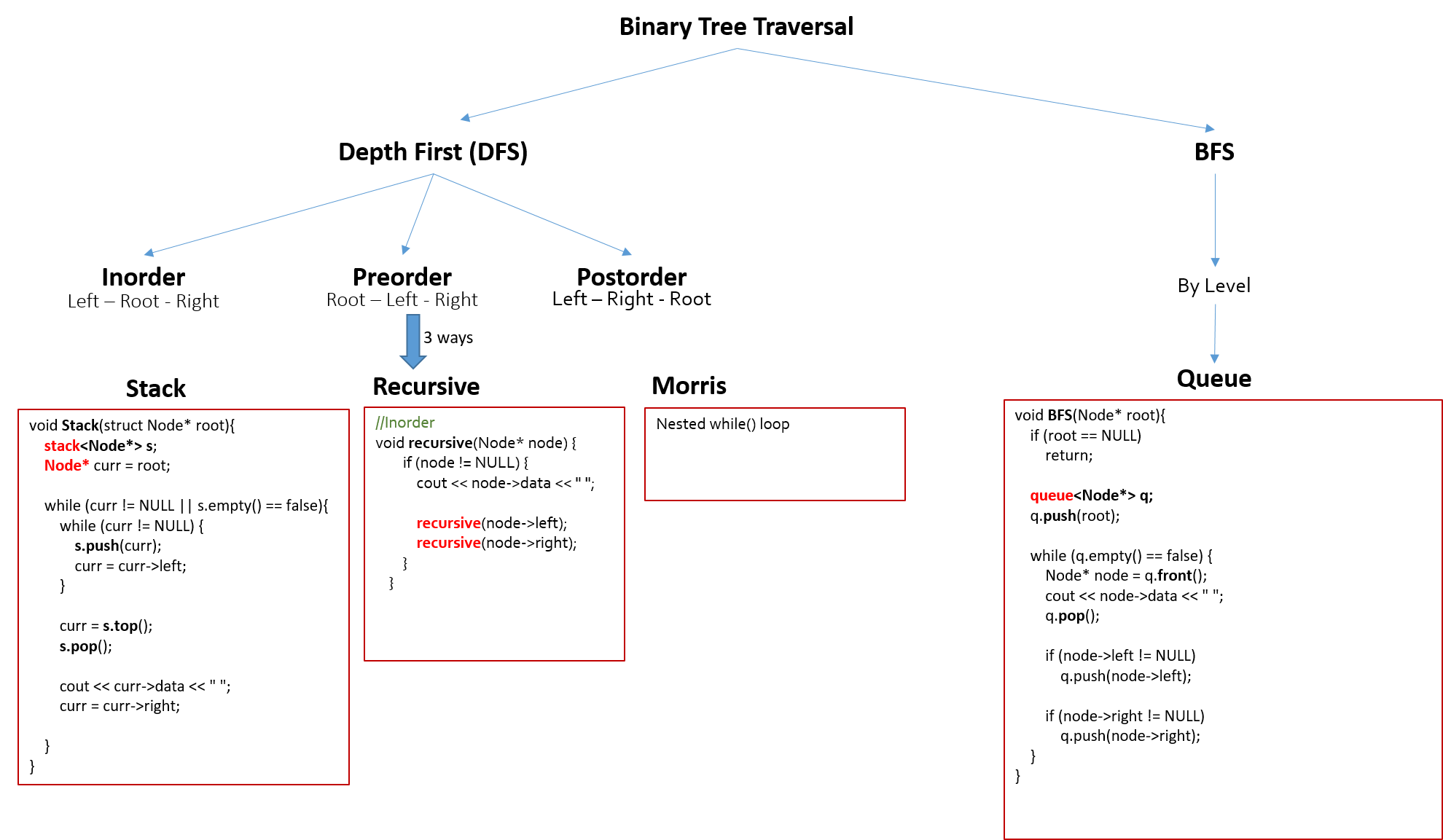
Element stored in binary tree & array

**a. Tree Traversal**

A tree can be traversed in many different way. The order is named by the position of ROOT in visiting priority.

|  |  |  |  |
| --- | --- | --- | --- |
| Order | Visit priority |  | Example |
| **Inorder** | Left - **Root** - Right | DFS | ABCDEFGHI |
| **Preorder** | **Root** - Left - Right | DFS | FBADCEGIH |
| **Postorder** | Left – Right - **Root** | DFS | ACEDBHIGF |
| **Level-order** | By level, from top to bottom | BFS | FBGADICEH |





1. **Operator**

**1. Operators**

**2. cmath()**

#include <cmath>

//a2 x squared pow(x, 2)  
double pow(double base, double exponent);

// square root x

float sqrt(float x);

//absolute(a)

double abs(double x);

//cosin(a)

float cos(float angle);

**is Prime Number?**

Số nguyên tố K: chia hết cho 1 và chính nó (has no dvisor other 1 and itself)  
 2 3 5 7 11  
  
 Tìm số n chia hết cho K từ 2 đến

|  |
| --- |
| //Write a for loop to look for a divisor from 2 till √n  bool isPrime(int n) {  for (int i = 2; i <= (int)(sqrt(n) + 1); i++) {  if (n % i == 0)  return false;  }  return true;  } |

**Greatest Common divisor**

Tìm **ước chung lớn nhất** (Số k lớn nhất mà A và B đều chia hết cho k)  
Lấy số lớn hơn chia cho số nhỏ hơn, tìm số dư (mod) rồi lặp lại cho đến khi chia hết

|  |  |
| --- | --- |
| **def** GCD(a, b) :  if (b == 0) :  return a;  return GCD(b, a % b);  if a > b:  GCD(a,b)  else:  GCD(a, b) | A = 20 , B = 15  Because A > B => call GCD(20,15)  B(15) != 0 => call GCD(15, 5)   B(5) != 0 => call GCD(5, 0)  B(0) == 0 => return A(**5**) DONE! |

**Least common Multiple**

Tìm **bội chung nhỏ nhất** (Số k nhỏ nhất mà k chia hết cho cả A và B)  
 LCM = AB / GCD(A,B)

|  |  |
| --- | --- |
| **def** LCM(a, b) :  return int((a \* b) / GCD(a, b)); | A = 20 , B = 15 Greatest common divisor = 5 => LCM = 20\*15 / 5 = **60** DONE! |

**Distance between 2 points**

A(x1, y1) and B (x2, y2)

|  |
| --- |
| float distance(int x1, int y1, int x2, int y2)  {  // Calculating distance  return sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2) \* 1.0);  } |

**Calculate Area**

|  |
| --- |
| **Triangle**  bool isValidTriangle(float a, float b, float c) {  if ((a + b <= c) || a + c <= b || b + c <= a)  return false;  return true;  }  float calArea(float a, float b, float c)  {  //Tổng 2 cạnh phải lớn hơn cạnh còn lại  if (!isValidTriangle(a, b, c))  return 0;  //Chu vi  float s = (a + b + c) / 2;  //Diện tích  return sqrt(s \* (s - a) \* (s - b) \* (s - c));  } |
| **Square**  int areaSquare(int side)  {  int area = side \* side;  return area;  } |
| **Circle**  #define PI 3.1459  float areaCircle(float r) {  //r : radius    //Chu vi = 2πr  float s = 2 \* r \* PI;  //A = πr2  float area = PI \* r \* r;  return area;  } |

1. **Recursive**

**Fibonacci Number**: f(n) = f(n – 1) + f(n-2)

|  |  |
| --- | --- |
| int fib1(int n) {  if (n == 0)  return 0;  if (n == 1)  return 1;  return fib(n - 1) + fib(n - 2);  } | //De quy co nho - hash map  unordered\_map<int, int> memo;    int fib(int n) {  if (memo.count(n) > 0)  return memo[n];  if (n == 0) {  memo[0] = 0;  return 0;  }  if (n == 1) {  memo[1] = 1;  return 1;  }  memo[n] = fib(n - 1) + fib(n - 2);  return memo[n];  } |

1. **Advanced Topics**
   1. **Difference Array**

* **To update value in range with O(1) time complexity.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Tạo 1 array chứa difference:   
Array A[7] =

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Difference D[7] =

2 types of queries:

* **update(left, right, x)**: Adds x to all values from A[left] to A[right]
* **printArr**(): print all

Thay vì chạy loop để update từng value từ left -> right. Tốn O(n) time

Update trên Difference Array sẽ tốn O(1) time.

* **update**(left, right, x):   
  **D[left] += x;  
  D[right +1] -= x;**
* **printArr**():   
  A[0] = D[0];

A[i] = D[i] + A[i-1];

**Example**: update(1, 3, 2)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 1 | 1 | -1 | 1 | 1 |

Difference D[7] =

D[left] = D[1] = 3;

D[right +1] = D[4] = -1;

printArr()

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 4 | 5 | 6 | 5 | 6 | 7 |

Array A[7] =

* 1. **2 Pointers**
* **Left-Right** pointers:Find a pair (2 elements) in array that satisfy a certain condition (sum up to K value)   
  **-> Find 2 sum**, **3 sum**.   
  -> Time O(nlogn)

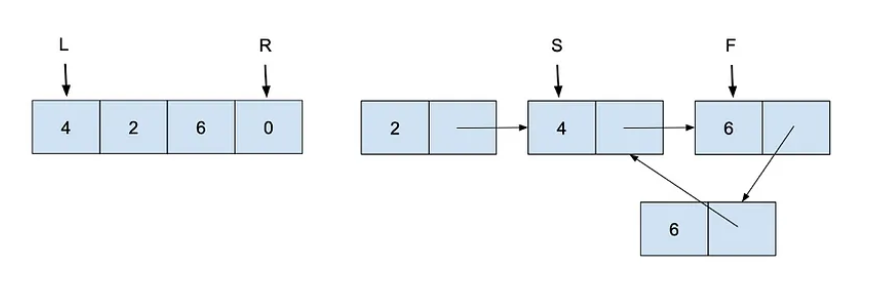
|  |
| --- |
| **sort**(arr, arr + length);  **sort**(vec.begin(), vec.end());  while (**left < right**) {  if (arr[left] + arr[right] == //...)  //...  if (//...)  **left++;**  else  **right--;**  } |

* **Reverse** : **Reverve array or string**   
  -> Time O(n)

|  |
| --- |
| void **reverse**(int start, int end, vector<int>& nums) {  while (start < end) {  **//swap(start-end)**  start++;  end--;  }  } |

* **Fast-slow** pointers: **Detect cycle.**   
  -> Time O(n)

|  |
| --- |
| ListNode\* slow = head;  ListNode\* fast = head;  while (**fast->next != NULL && fast->next->next != NULL**) {  **fast = fast->next->next;**  **slow = slow->next;**  //Has cycle, return the node where fast & slow meet  if (fast == slow)  return slow;  } |



* **Find Max area**: [Container with most water (leetcode)](https://leetcode.com/problems/container-with-most-water/description/)  
  -> Time O(n)

|  |
| --- |
| while (**left < right**) {  if (height[left] > height[right]) {  **water = height[right] \* (right - left);**  right--;  }  else {  **water = height[left] \* (right - left);**  left++;  }  **max = (max > water) ? max : water;**  }  return max; |

* **Trapped water**: [**Trapping rain water** (leetcode)](https://leetcode.com/problems/trapping-rain-water)  
  -> Time O(n)

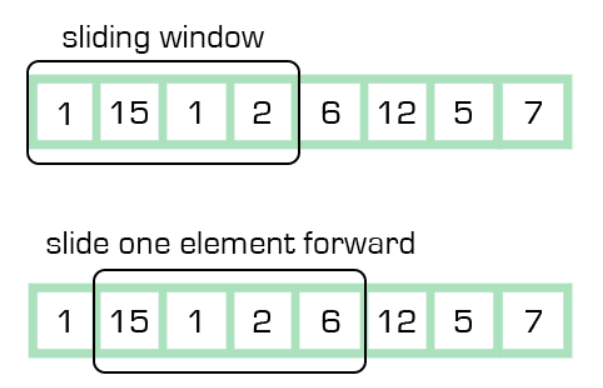
Use 2 pointers left-right to find L\_max & R\_max.   
Calculate trapped water based on L\_max & R\_max.

|  |
| --- |
| int **left** = 0, **right** = height.size() - 1;  int **L\_max** = height[left];  int **R\_max** = height[right];  int sum = 0;  while (**left < right**) {  L\_max = max(L\_max, height[left]);  R\_max = max(R\_max, height[right]);  if (height[left] >= height[right]) {  if (height[right] < R\_max)  **sum += (R\_max - height[right]);**  right--;  }  else {  if (height[left] < L\_max)  **sum += (L\_max - height[left]);**  left++;  }  } |

* 1. **Sliding Window**

Find **Sub Array / Sub String** with **fixed/ unfixed** Window size

* **Max/Min sum sub array**: Max/min sum subarray of size k  
  -> Time O(n)



* 1. **Prefix Sum**
  2. **Basic Math: Sieve Algorithm & Inverse Modular**

abc

* 1. **Advanced String: KMP & Rolling Hash**

abc

* 1. **Monotonic Stack**

Monotonic Stack: Stack tăng dần hoặc giảm dần.

* 1. **Union Set**

abc

* 1. **Bitmask DP**

abc

* 1. **Linked List – Floyd (Slow - Fast) / Detect cycle**

abc

* 1. **BIT Tree / Fenwick Tree**

abc

* 1. **Segment Tree**

abc

* 1. **Graph: Minimal Spanning Tree**

abc

* 1. **Trie**

abc

* 1. **Bit Manipulation**

abc

1. **Template**

|  |
| --- |
| // function template II  #include <iostream>  using namespace std;  template <class T>  T GetMax (T a, T b) {  return (a>b?a:b);  }  int main () {  int i=5, j=6, k;  long l=10, m=5, n;    k=GetMax(i,j);  n=GetMax(l,m);  //or  k=GetMax<int>(i,j);  n=GetMax<long>(l,m);  cout << k << endl;  cout << n << endl;  return 0;  } |

1. **Implement string class**

|  |
| --- |
| // C++ program to illustrate the  // above discussed functionality  #include <cstring>  #include <iostream>  using namespace std;    // Custom string class  class Mystring {        // Initialise the char array      char\* str;    public:      // No arguments Constructor      Mystring();        // Constructor with 1 arguments      Mystring(char\* val);        // Copy Constructor      Mystring(const Mystring& source);        // Move Constructor      Mystring(Mystring&& source);        // Destructor      ~Mystring() { delete str; }  };    // Function to illustrate Constructor  // with no arguments  Mystring::Mystring()      : str{ nullptr }  {      str = new char[1];      str[0] = '\0';  }    // Function to illustrate Constructor  // with one arguments  Mystring::Mystring(char\* val)  {      if (val == nullptr) {          str = new char[1];          str[0] = '\0';      }        else {            str = new char[strlen(val) + 1];            // Copy character of val[]          // using strcpy          strcpy(str, val);            cout << "The string passed is: "               << str << endl;      }  }    // Function to illustrate  // Copy Constructor  Mystring::Mystring(const Mystring& source)  {      str = new char[strlen(source.str) + 1];      strcpy(str, source.str);  }    // Function to illustrate  // Move Constructor  Mystring::Mystring(Mystring&& source)  {      str = source.str;      source.str = nullptr;  }    // Driver Code  int main()  {      // Constructor with no arguments      Mystring a;        // Convert string literal to      // char array      char temp[] = "Hello";        // Constructor with one argument      Mystring b{ temp };        // Copy constructor      Mystring c{ a };        char temp1[] = "World";        // One arg constructor called,      // then the move constructor      Mystring d{ Mystring{ temp } };      return 0;  } |

**Thread**

Class to represent individual *threads of execution*.  
  
A *thread of execution* is a sequence of instructions that can be executed concurrently with other such sequences in *multithreading* environments, while sharing a same address space.

An initialized [thread](https://www.cplusplus.com/thread) object represents an active thread of execution; Such a [thread](https://www.cplusplus.com/thread) object is [*joinable*](https://www.cplusplus.com/thread::joinable), and has a unique [*thread id*](https://www.cplusplus.com/thread::get_id).